Project Number	AC026
Project Title	Secure Electronic MarketPlace for Europe SEMPER
Deliverable Security Class	Public
CEC Deliverable Number	AC026/SMP/CT2/DS/P/014/b1
Contractual Date of Delivery	October 31, 1996
Actual date of Delivery	November 22, 1996
Title of Deliverable	Deliverable D14
	Architecture of Payment Gateway
Contributing Workpackages	WP21
Nature of the Deliverable	Report
Author(s)	B. Baum-Waidner (R3), A. Mueller (R3), T. Pedersen (Cryptomathic), M. Schunter (Univ. Dortmund)
Editor(s)	Armin Müller (R3)
Keywords	Architecture, Payment Gateway, Payment Protocol

SEMPER

SEMPER Consortium:

Architecture of Payment Gateway

Deliverable D14 of ACTS Project AC026

Public Report
November 22, 1996

© SEMPER Consortium, represented by IBM France, La Gaude November 22, 1996.

SEMPER is part of the European Commission's ACTS Programme (Advanced Communications Technologies and Services). Funding is provided by the partner organisations, the European Union, and the Swiss Federal Department for Education and Science.

The members of the SEMPER consortium are Cryptomathic (DK), DigiCash (NL), EUROCOM EXPERTISE (GR), Europay International (B), FINTEL (B), FOGRA Forschungsgesellschaft Druck (D), GMD - German National Research Center for Information Technology (D), IBM La Gaude (F), IBM Zurich Research Lab (CH), INTRACOM (GR), KPN Research (NL), Otto-Versand (D), RegionLink (B), r3 security engineering (CH), SEPT (F), Stichting Mathematisch Centrum / CWI (NL), Systems Synthesis Ltd. (UK), University of Freiburg (D), University of Dortmund (D). Banksys (B), Banque Générale du Luxembourg (L) and Telekurs (CH) are associated with SEMPER.

For more information on SEMPER visit

<http://www.semper.org>

or contact

Gerard Lacoste	Michael Waidner
IBM La Gaude	IBM Research Division
Centre d'Etudes et Recherches	Zurich Research Laboratory
Le Plan du Bois	Säumerstrasse 4
F-06610 La Gaude, France	CH-8803 Rüschlikon, Switzerland
Phone + 33 92 11 48 07	Phone +41 1 724 8220
Fax +33 93 24 45 45	Fax +41 1 710 3608
Mail lacoste@vnet.ibm.com	Mail wmi@zurich.ibm.com

Management Summary

This report is a public deliverable of *SEMPER*, “Secure Electronic Marketplace for Europe”. The main objective of *SEMPER* is to develop, implement, trial and evaluate an open security architecture for electronic commerce over open networks such as the Internet.

While the initial *SEMPER* architecture is described in deliverable D03 (“Basic Services, Architecture and Design”), this deliverable gives a generic system architecture for a payment gateway to mediate between closed financial networks and open data networks. In order to enable a generic architecture, all relevant payment systems and transactions were identified on the open networks as well as on the financial networks. An architecture is proposed to support those systems and transactions and the addition of further services later.

The resulting payment gateway will represent the bank towards the players on the open network, and the players on the open network towards the banking network.

The proposed architecture is compliant with the *SEMPER* architecture and therefore independent of specific payment systems: any payment system that can be used in *SEMPER* is also supported by the gateway. Transactions like authorisation and capture for credit-like payment systems, withdrawal of electronic cash for cash-like systems and depositing of electronic checks for electronic-check models and many other transactions will be performed in a secure way. Banks will be enabled to offer all Internet payment instruments by just providing a single interface to their bank network. An appropriate firewall concept will be used to connect the payment gateway to the open network.

The proposed architecture provides interoperability between various Internet payment protocols and the corresponding protocols of the bank networks in a most efficient and flexible way.

Table of Contents

1. INTRODUCTION 4

2. PAYMENT MODELS AND PROTOCOLS 6

2.1 Credit-like Model 6
2.1.1 Description of the Payment Model 6
2.1.2 Existing Internet Payment Systems 6
2.1.3 Bank Protocols 9

2.2 Cash-like Model 11
2.2.1 Opening Account 11
2.2.2 Withdrawal 13
2.2.3 Payment 13
2.2.4 Deposit 14
2.2.5 Possible Additional Requirements to the Gateway 16

2.3 Electronic-check Model 16
2.3.1 Getting Checks 17
2.3.2 Deposit 17
2.3.3 Additional Requirements to the Gateway 17

2.4 Electronic Fund Transfer 18

3. ARCHITECTURE OF THE PAYMENT GATEWAY 19

3.1 Protocol Mapping 19

3.2 Intermediate Protocols 20

3.3 Architecture 21

3.4 The Payment Service Block 23
3.4.1 Alarm Clock Service and Time-outs 23
3.4.2 Other Payment Gateway Internal Services 23
3.4.3 Transaction-ID 24
3.4.4 Protocol and Mapping 24
3.4.5 Transaction State Storage and Input Queues 30
3.4.6 Getting Input Events by Function Get_map 30
3.4.7 Processing of Events 31
3.4.8 Additional Payment Systems 32
3.4.9 Scalability 32
3.4.10 Fault Tolerance 32
3.4.11 Flow Diagram 33
3.4.12 Advantages of this Approach 34

4. CONNECTION TO THE INTERNET 35

5. CONCLUSIONS 36

5.1 Summary 36

5.2 The Next Steps 36

6. REFERENCES 37

7. APPENDIX: GLOSSARY 38

Table of Figures

FIGURE 1: THE PAYMENT GATEWAY PROVIDES INTEROPERABILITY BETWEEN INTERNET PAYMENT INSTRUMENTS AND BANK NETWORK PROTOCOLS. 4
FIGURE 2: BASIC PURCHASE FLOW OF THE SET PROTOCOL. 7
FIGURE 3: POSSIBLE SEQUENCES OF TRANSACTIONS (ISO 8583). 10
FIGURE 4: OPENING ACCOUNT. 12
FIGURE 5: WITHDRAWAL OF ECASH®. 13
FIGURE 6: PAYMENT. 14
FIGURE 7: DEPOSIT. 15
FIGURE 8: PAYMENT GATEWAY MAPS BETWEEN PROTOCOLS USING *SEMPER* CONTAINERS WITH EDIFACT MESSAGES AND BANK PROTOCOLS. 18
FIGURE 9: MAPPING CREDIT-LIKE PROTOCOLS. 20
FIGURE 10: THE INTERMEDIATE PROTOCOL ALLOWS $N + M$ MAPPINGS INSTEAD OF N X M MAPPINGS. 20
FIGURE 11: ARCHITECTURE OF PAYMENT GATEWAY. 22
FIGURE 12: FLOW DIAGRAM OF PAYMENT SERVICE BLOCK. 34
FIGURE 13: A FIREWALL NEEDS TO PROTECT THE PAYMENT GATEWAY FROM ATTACKERS ON THE INTERNET. 35

1. Introduction

Customers, merchants and financial institutions are players in payment-related business applications over the Internet, using, e.g., *SEMPER* compliant software. To connect the existing traditional payment infrastructure of the financial institutions to the Internet, a payment gateway is needed. The payment gateway maps the protocols between the world of the Internet and the existing financial networks of the banks (see Figure 1).

A gateway is under the control of either one bank or a financial institution representing several banks. The gateway is connected to the existing payment infrastructure of those banks, in their roles as acquirers or issuers, respectively, depending on the kind of transaction.

The payment gateway communicates with the bank networks that run in a trusted environment, from the bank's point of view. However, the Internet runs in an insecure and therefore highly untrusted environment from everybody's point of view. To secure the communication as well as complete business actions over the Internet, complex protocols as well as lower-layer supporting cryptographic operations (e.g. encryption, decryption, calculation and verification of digital signatures) are needed.

There are some well defined Internet payment protocols [JW95] for credit-like transactions (SET, iKP, etc.) and cash-like transactions (Ecash®, etc.). For the communication of the payment gateway with the banks many different protocols are used. For example, in Switzerland there is an acquirer system using a protocol called CAR (Credit Card Authorisation Request) to authorise and CDS (Credit Card Data Submission) to capture credit-card payments. Other countries use other protocols. The standard ISO 8583 [ISO93] originally defined as a bank-to-bank protocol, i.e., between the acquirer and issuer, can also be used between the gateway and bank instead of protocols like CAR and CDS.

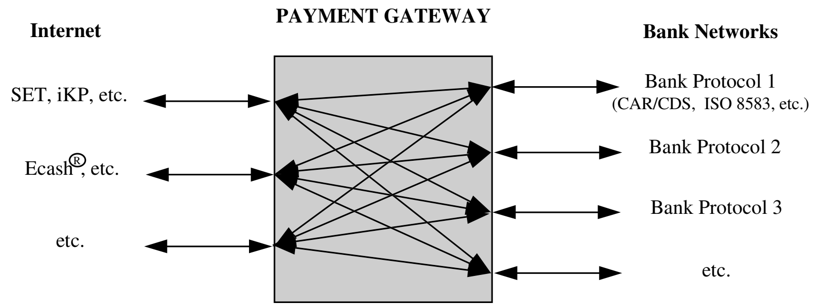

Figure 1: The payment gateway provides interoperability between Internet payment instruments and bank network protocols.

Towards the Internet, the gateway acts as an aquirer or an issuer for the various Internet payment systems. Towards the banking world it acts as a merchant or a customer 'speaking' the appropriate bank protocols.

Just as *SEMPER* is designed to support different existing and future payment systems, it is an essential goal of the payment gateway architecture to handle many different Internet payment instruments and bank protocols.

This report is structured as follows: Section 1 motivates the demand for a generic payment gateway. Section 2 identifies relevant payment models and protocols to be covered by the payment gateway. Section 3 proposes an architecture for a generic payment gateway which enables an efficient realisation, extendability, scalability due to a highly modular *SEMPER*-compliant concept. Section 4 suggests how to connect the payment gateway to the Internet securely. Section 5 gives the conclusions including the next steps for the payment gateway. In the Appendix, a glossary defining the mostly used terms in *SEMPER* is given.

2. Payment Models and Protocols

The current approach of the payment gateway supports four different models or classes of payment transaction protocols which we believe cover all reasonable classes currently available or to be expected in the future [see PAW96]:

- credit-like
- cash-like
- electronic check (e-check)
- electronic fund transfer (EFT).

Below we sketch the protocols for the different models by describing existing payment systems (SET, Ecash®) in more detail in order to understand which functionality is required of the gateway. The sketches are, to the extent possible, based on existing payment systems. However, as the aim is to define a generic payment gateway, we do not describe a single payment system, but try to generalise the descriptions in order to capture possible additional requirements of the gateway. Both the gateway to the issuer and to the acquirer are, of course, considered.

2.1 Credit-like Model

2.1.1 Description of the Payment Model

Credit-like payment systems are the electronic counterpart of the traditional credit card payment. The most important transactions involving the gateway are:

- **authorisation:** when a payee obtains authorisation, payment for the authorised amount is guaranteed.
- **capture:** completes the payment and results in the actual charge to the payer's account.

The next sections sketch the protocols for the Internet and the bank networks to illustrate the requirements to the gateway.

2.1.2 Existing Internet Payment Systems

For the credit-like payment model, payment systems like SET [MV96] and iKP [BGHH95] exist. These are well defined protocols to handle credit-card payments over the Internet. Figure 2 illustrates the general flow for the SET protocol. The flow for iKP is very similar. Please note that we do not aim at presenting the protocols themselves but just provide an impression of the flow of protocols supporting such payment systems.

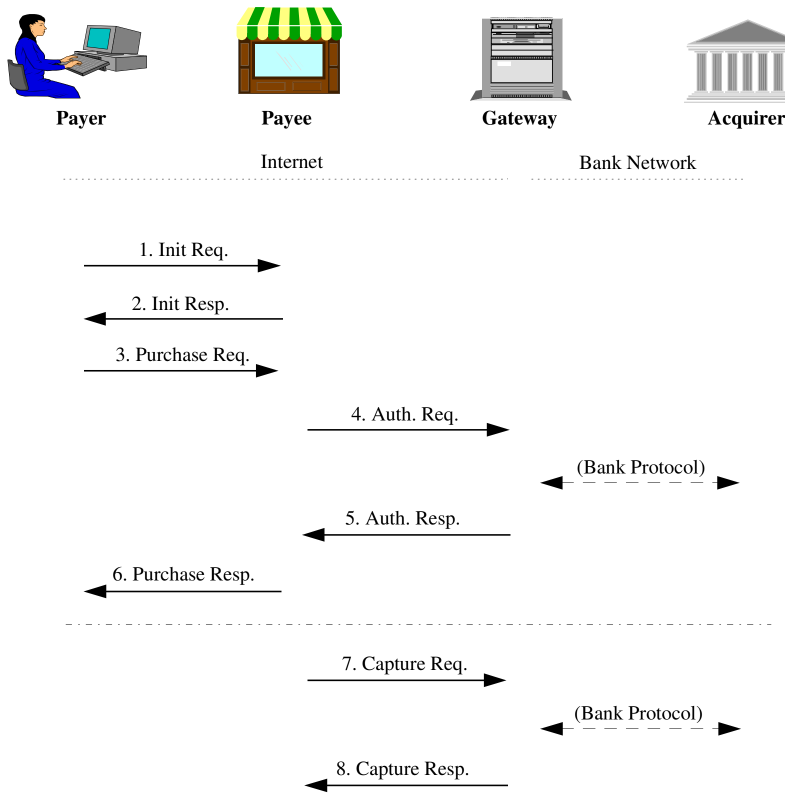

Figure 2: Basic purchase flow of the SET protocol.

The **initialisation** request (Step 1) and response (Step 2) provide a number of initialisation services between the payer and the payee. They provide both cryptographic initialisation, certificate exchange, and initialisation of transaction identifiers.

The **purchase** request (Step 3) and response (Step 6) messages are at the heart of the payment protocol. This is the payer/payee payment pair that embodies the payment from the payer's point of view. These messages provide the reference to the authorisation and price, and confirm the completion of the transaction. The purchase response message may be returned in different stages of the protocol. The information returned will depend on the stage of the protocol at which the purchase response is returned.

In Step 4 the payment gateway receives the **authorisation** request from the payee. The gateway will use the bank protocol to get an authorisation response from the acquirer (out of scope of the SET protocol). Upon receiving an authorisation response from the acquirer, the payment gateway generates the authorisation response (Step 5) for the payee (this is done within the scope of mapping to be performed by the payment gateway). Authorisation

provides the payee approval by the acquirer to continue processing, i.e. payment for the authorised amount is guaranteed.

In Step 7 the payee requests **capture**. (There may be a significant time lapse between the message requesting authorisation and the message requesting the capture). The gateway itself uses the bank protocol to get a capture response (out of scope of the SET protocol) and generates a capture response for the payee (Step 8). This activity completes the purchase for the gateway, and results in the actual charge to the payer’s account.

The capture could optionally be communicated by other means than the Internet.

Please note:

a) The SET protocol also includes a sales transaction that allows a payee to request authorisation and capture. The gateway may map such a SET transaction to *several* transactions of the bank protocol.

b) While the flow described here contains only a single capture request, the payee software is permitted to batch multiple requests into a single transaction. In the gateway this may result in a *sequence* of transactions for the bank protocol.

There are additional messages (not depicted in Figure 2) defined in the SET specifications:

The **inquire** request and response message pair enables the payer to inquire the status of the transaction. These can be sent any time after the payer has sent a message to or received a message from the payee.

Authorisation reversal request and response are used when authorisations need to be reversed. An authorisation reversal can be initiated by the merchant to change the amount of authorisation. It may be sent any time after the authorisation but before capture has been requested.

Capture reversal request and response are used for the capture reversal process, i.e., to reverse a capture before settlement (the actual crediting to the payee's account) has occurred.

Credit request and response are used for the credit process. It is similar to the capture reversal, except that it is used after the settlement.

Gateway certificate request and response provide a mechanism for the payee to obtain the gateway certificates (for encryption) which he needs before he can encrypt a message to the gateway.

Batch administration request and response can be used if a batch of capture tokens has been sent to the payment gateway to be processed together. The batch administration messages provide the means for the payee to open, close, and obtain status and transaction data on batches maintained by the payment gateway for managing the payee’s transactions.

A general **error message** is provided for reporting protocol and/or processing errors.

Most of the Internet protocol steps require cryptographic operations like encryption/decryption and calculation/verification of signatures [STIN95]. The SET protocol relies on certificates for the authentication of the payer, payee and gateway. The certificates are linked in a hierarchical public key infrastructure. Before a public key is used, the validity of the corresponding certificate is checked against a trusted public key (possibly by going through a chain of certificates).

The certificates are transferred and exchanged in the transactions of the basic flow given in Figure 2. At the current level of abstraction we are not considering the flow of certificates in more details, e.g. we do not describe any extra steps needed for the certificate verification.

2.1.3 Bank Protocols

The bank protocols are typically different for every acquirer. We sketch *an example* of such a bank protocol that is used in Switzerland for the communication of point of sales (POS terminals) with the credit-card associations. This protocol can also be used for the payment gateway. The protocol for authorisation is called CAR 'credit card authorisation request' and for capture CDS 'credit card data submission'.

In addition we sketch the ISO 8583 protocol specifically designed for the communication between acquirer and issuer, but it is also a possible means for the communication between the acquirer and gateway.

2.1.3.1 Authorisation and Capture with CAR and CDS

The specifications of CAR 'credit card authorisation request' define just the format for the authorisation request and response messages that can be used to communicate with the bank network. The messages for the different credit-card associations are very similar but may differ in the number of fields and their sizes. The specifications do not describe a detailed protocol flow, e.g., time-outs, retries and error messages.

The CAR protocol can not be used alone. A separate protocol is needed for the capture. The specifications of CDS 'credit card data submission' require that certain fields of the authorisation requests and responses have to be stored to create afterwards a file that is transferred to the bank for capture. For instance we need from the authorisation response the amount authorised and the authorisation number. The file typically contains the capture requests for many transactions collected over some period of time.

2.1.3.2 ISO 8583

The standard ISO 8583 [ISO93] originally defined as a bank-to-bank protocol, i.e., between the acquirer and issuer, can also be used between the gateway and bank instead of protocols like CAR and CDS. The messages for the ISO 8583 protocol can be constructed in a very flexible way. They include:

message type identifier:

- version number
- message class: one of authorisation, financial, reversal/chargeback, file action, reconciliation, fee collection, network management
- message function: one of request, request response, advice, advice response, notification
- transaction originator: one of acquirer, acquirer repeat, card issuer, card issuer repeat, other, other repeat.

one or two bitmaps:

- a series of 64 (or 128) bits is used to identify the presence (denoted by 1) or absence (denoted by 0) of each data element in the message. For instance the second bit indicates with a value of 1 the presence of the data element with the primary account number.

data elements:

- data elements are present or absent as indicated with the individual bits in the bitmap.

The message flow is specified as 'pairs' of requests and responses for authorisation, financial transactions, file action transactions, reversal transactions, chargeback transactions etc. If the original request is not responded to for whatever reason, the sender shall send a reversal. If the message cannot be identified the receiver may send a message indicating inability to parse the original message.

The following figure shows possible sequences of transactions:

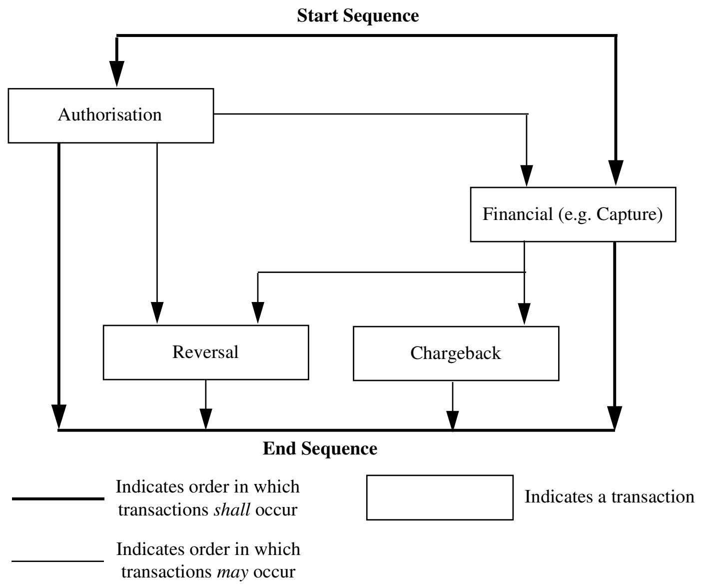

Figure 3: Possible sequences of transactions (ISO 8583).

A financial message can contain the capture for one transaction only. For the gateway this may mean after receiving a SET Internet capture request including a collection of capture tokens that the gateway has to send a sequence of capture messages over the bank network and perform a sequence of state transitions.

2.2 Cash-like Model

Cash-like payment is the electronic counterpart of normal cash payment. Such a payment system typically needs the following five transactions:

- **opening account**: Establish an account for electronic cash
- **withdrawal**: The payer gets electronic cash from an issuer
- **payment**: The payer pays the payee
- **deposit**: The payee deposits the received cash at its acquirer (the payer can deposit unused cash at its own bank)
- **settling**: The issuer and acquirer settle their accounts.

The last transaction is internal for the banks and will not be considered further here, as it does not involve the gateway. A payment gateway will be involved in the second and fourth transaction as a gateway to the issuer and acquirer, respectively. Some electronic cash systems also provide an on-line method for opening an account. Thus the payment gateway should also be able to support this transaction. Depending on the payment system, deposit or withdrawal may be done as part of payment and thus involve the gateway.

In order to illustrate the requirements to the gateway imposed by cash-like payment systems we describe possible message flows in protocols for opening account, withdrawal, payment and deposit. This is inspired by the typical and well-known representative Ecash® [DIGI96] but the presentation tries to illustrate possible variations required by other payment systems (e.g., Clip).

2.2.1 Opening Account

This may or may not be done on-line. In Ecash® an on-line protocol for opening accounts is available, see Figure 4. The protocol assumes that the payer already has an account with a bank (issuer) that provides Ecash®.

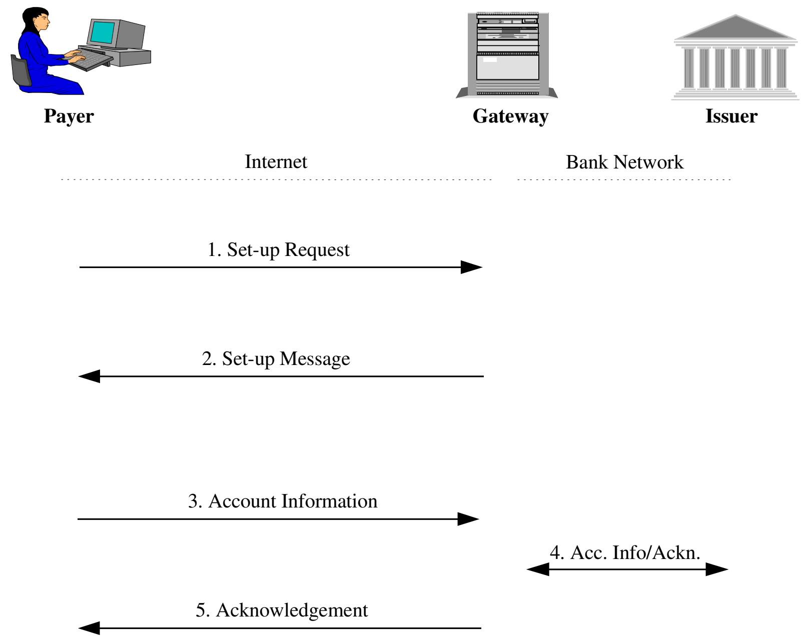

Figure 4: Opening account.

The protocol starts with a set-up request (Step 1) sent from the payer to the gateway. The gateway returns the set-up message to the payer (Step 2). This message is signed with the set-up secret key of the gateway. The corresponding set-up public key must be known to the payer beforehand (e.g., the payer's software comes with the set-up public key). The set-up message contains the issuer's public key. It is important that this key is authentic, as all issuer signatures will be verified against it (if the account is opened with a fake set-up public key to a fake bank resulting in a fake issuer's public key, than the subsequent operations could appear as valid). The set-up request and message are not encrypted.

After that, the payer generates a private/public key pair for signatures, and sends to the gateway the new public key, account number, email address and possibly other information (Step 3). The gateway verifies the message and forwards a request to open an Ecash® account to the bank. This request must contain the account number. As a result of processing this request, the bank either returns a positive or negative answer. If the answer is positive the gateway sends to the payer a signed acknowledgement, which for Ecash® contains the account number and all the relevant coin types (needed to verify coin signatures or to know when coins expire).

2.2.2 Withdrawal

Next the payer can withdraw Ecash® using the protocol shown in Figure 5:

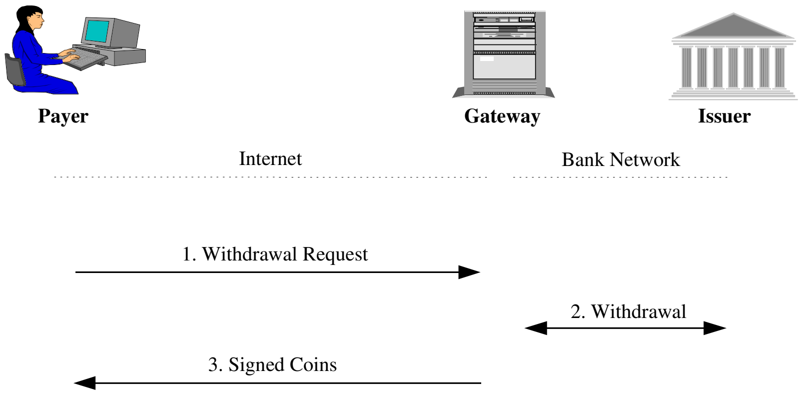

Figure 5: Withdrawal of Ecash®.

The flow in this three step protocol seems to be quite representative for withdrawal of electronic cash. In more detail the Ecash® protocol works as follows. A withdrawal starts with the payer choosing the random numbers for the coins he wants to withdraw. These coin numbers are then ‘blinded’ (i.e., hidden so that the gateway cannot learn the coin numbers of the actual coins obtained by the payer) and the whole message is signed by the payer before it is sent to the gateway (Step 1). The gateway verifies the signature and forwards a request to withdraw the coins to the bank. If the bank allows the withdrawal it withdraws the amount from the payer’s account and instructs the gateway to send the signed coins to the payer. The payer finally obtains the coins to be used by removing the blinding factors, which were added in Step 1.

A remark on the minting process is necessary here. In the above description the coins are signed ('minted') in the gateway. Obviously this requires that the part of the gateway doing this is placed physically secure, so that the bank has full control over this process. Alternatively, it could be argued that the minting process should be done by the bank outside of the gateway. However, this would require that the details on the messages of the Internet cash protocol should be known on the banking network 'behind' the gateway. That would contradict one of the goals of the gateway.

2.2.3 Payment

A payment is a transaction between a payer and a payee. In on-line systems (such as Ecash®) deposit may take place during the payment. However, for the sake of generality we describe payment independently of deposit. Communication during a successful payment is shown in Figure 6.

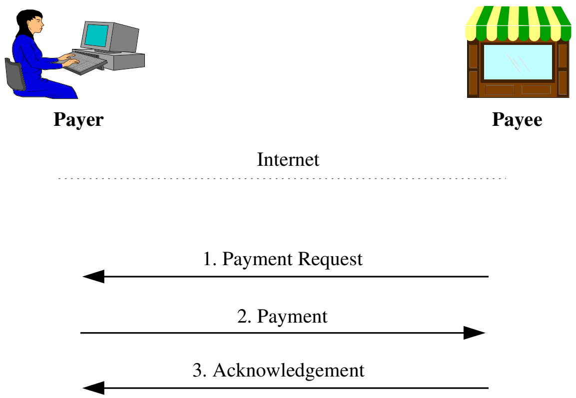

Figure 6: Payment.

The meaning of these messages should be clear. In some cases Steps 1 and 2 may be repeated several times before a final acknowledgement is sent from the payee (e.g., in case of micro-payments). In that case the payment request for each part of the payment may look different from each other.

It is also possible that a payment is initiated by a payer, rather than the payee. In that case the payment request message is omitted, and the payer will have to derive the information normally received in the payment request from the context of the transaction.

2.2.4 Deposit

The deposit protocol allows the payee to deposit received electronic cash at the gateway of the acquirer. As part of the deposit protocol the gateway may detect (or be notified by a double-spending facility) that the coins have been deposited before. This double-spending may either be due to the payer having cheated and spent the cash more than once, or the payee trying to deposit the same money more than once.

The detection of double-spending requires that a double-spending facility with a database of all spent coin numbers is available to the gateway. This also means that in every deposit tracing information (e.g., unique coin numbers as in Ecash®) must be forwarded from the gateway to the double-spending facility. This double-spending machine can, e.g., be an internal component of the gateway or be set-up on the Internet to be accessible by different gateways.

The information exchanged in the deposit protocol depends on how such frauds are coped with. Before describing how Ecash® deals with deposit, two alternatives for deposit are discussed.

2.2.4.1 Deposit 1

The payee sends a subset of the received messages (coins) to the gateway (Step 1 in Figure 7). The gateway verifies the messages. If the gateway accepts the payment, it sends a signed message to the payee and requests him to acknowledge the deposit message (Step 2). Assuming the payee sends back the signed deposit message (unique for this transaction, Step 3) the gateway verifies the signed message and forwards the result to the acquirer (Step 4), who updates the payee's account and sends an acknowledgement back to the payee through the gateway (Step 5). A gateway not accepting the deposit must provide in Step 2 a signed message from a previous deposit as a proof of hat deposit.

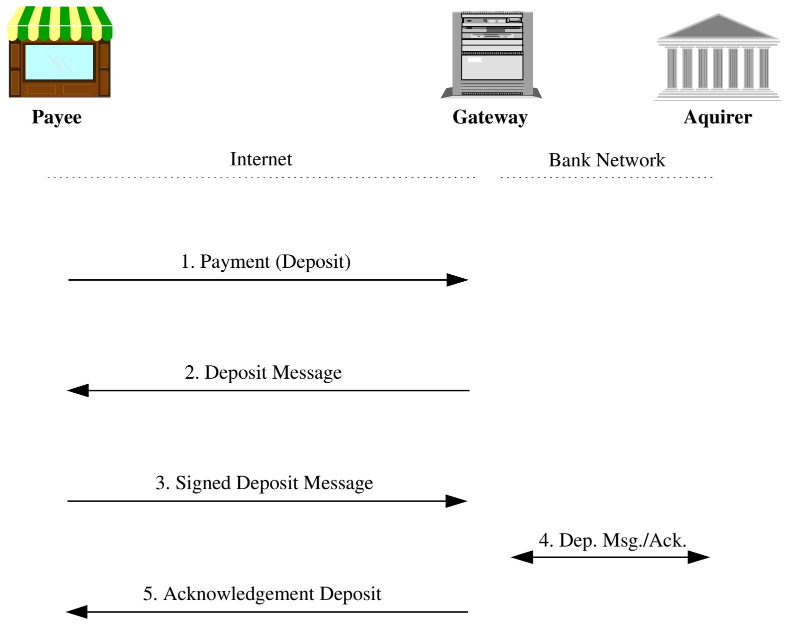

Figure 7**: Deposit.**

2.2.4.2 Deposit 2

This variant also follows the five step deposit depicted in Figure 7. Assume that as part of the payment protocol the payee inserts a hash value. A pre-image of this hash-value will serve as the signature on the signed deposit message in Step 3. More precisely, the following will take place.

If the gateway accepts the payment, the payee is asked to provide a pre-image of the hash value in Step 3. If the gateway does not accept the deposit given the initial message from the payee, it must provide a pre-image of the hash value as a proof of a previous deposit. The payee must then provide his own pre-image. A pre-image different from that of the gateway

will count as a proof that the payee has not deposited the money previously. If the hash function is collision resistant the payee cannot come up with different pre-images in two different executions of deposit.

2.2.4.3 Ecash®: Combined Deposit and Payment

In case deposit takes place during payment, it would naturally be done between Steps 2 and 3 of the payment protocol (see Figure 6). This is done in Ecash®, where payment is combined with a three step deposit protocol as follows:

In Step 1 of the payment the payer receives a payment request including the payee's bank number. The payer selects a subset of his coins with a sum equal to the required amount. The payer commits to some random number by calculating the hash value of it. The hash value will be sent to the payee. In a dispute, only the payer that indeed made the payment will be able to open the commitment to the payee by revealing the input to the hash function. (It is important to be able to prove this, since Ecash® payments are anonymous.) Next, the payer generates a payment header which contains information about the recipient of the payment (due to anonymity it does not contain any information about the payer).

The payee sends the payment message to the gateway (Step 1 of Figure 7), which verifies it. If the coins have not been spent before, they are added to the database of spent coins, the payee's account balance is increased by the acquirer and an acknowledgement is sent to the payee. This acknowledgement is signed by the gateway (using the secret key of the acquirer). After receiving the acknowledgement, the payee sends an acknowledgement to the payer (Step 3 of payment).

2.2.5 Possible Additional Requirements to the Gateway

In addition to the (basic) protocols described above the gateway may be involved in other types of transactions. Three examples are:

- In Ecash® the payer can only deposit his coins by making a payment to his bank account. Thus the gateway (and the issuer) should be able to support the role of the acquirer.
- It may not be feasible to make a large number of small payments in an on-line system, if each micro-payment requires a connection to the issuer or acquirer. In order to circumvent this problem an authorisation protocol, which enables the payee to accept a large number of payments up to a certain amount off-line, may be offered. If that is the case the gateway must be able to handle such a protocol.
- The gateway may need to support recovery transactions in case a payer loses his money.

2.3 Electronic-check Model

An electronic check payment system is the electronic counterpart of normal check payments. The typical transactions are:

- **payment**: the payer signs a check and sends it to the payee;
- **deposit**: the payee sends the check to its bank for deposit;
- **clearing**: the acquiring bank clears the check with the issuing bank.

The most well-known check payment system is probably the FSTC electronic check payment system [FSTC95] which also defines the protocols mentioned above. We are not aware of any existing check system with public details on the actual message flow. Thus, presently, it is not possible to describe the message flow or the inner working of a check system in detail. However, for the payment protocol it is reasonable to expect a two or three step flow as for cash-like payments (see Figure 6).

The gateway can be expected to be involved in two protocols:

- the payer needs a gateway to the issuer in order to get new cheques and
- the payee needs a gateway to the acquirer in order to deposit received checks.

2.3.1 Getting Checks

The payment gateway is needed when the payer needs more checks. In practice an electronic check can be expected to be a signed message stating the validity of a check. A special protocol may be required the very first time a person wants to register as a payer of the check system (as the protocol for opening an electronic cash account, see Figure 4). The actual protocol flow when registering initially (if needed) and for getting checks will most likely be as shown in Figure 4 and Figure 5, respectively.

2.3.2 Deposit

A received check will be deposited by the payee endorsing the checks and sending it to the gateway. The gateway will check the signatures on the check and forward a message to the bank with the relevant information on the payer, payee and amount. Most likely, the protocol for this will resemble the deposit protocol for cash. Thus the gateway should be prepared to support such transactions.

If the check system is off-line (as FSTC), checks will be deposited in batches. Thus the gateway should also be able to handle this situation.

2.3.3 Additional Requirements to the Gateway

In order to support a check like payment system, the gateway may have to handle other transactions such as revocation of certificates (e.g., in case a secret key of the payer has been compromised).

The FSTC project also promises 3 other possible message flows. One is simply electronic funds transfer, where the payer asks his bank (through the gateway) to transfer the specified amount to the acquirer. The two remaining flows allow the payee to contact the payer’s bank (“Z flow”, see [FSTC95]) and to let the payer contact the payee’s bank (the “Lockbox flow”, see [FSTC95]). Without knowing the details it is hard to say what will be required from the gateway in order to support this. It does seem, however, that the EFT payment will be covered by the gateway as described in the next section.

2.4 Electronic Fund Transfer

Electronic fund transfer (EFT) is the electronic counterpart of conventional fund transfer using paper. EFT means that a user - an individual or a business - can send a payment order from his/her computer to the bank to transfer money from one account to another resulting in a direct deposit to the payee's bank account. In addition to EFT it may be very useful to provide account information.

Money transfer services require a high level of security, especially non-repudiation. For EFT the following security services are required:

- confidentiality
- integrity
- authentication
- non-repudiation.

There are no standard protocols for EFT to communicate between the user and the bank, but many country specific EFT protocols.

The planned usage of UN/EDIFACT (ISO 9735) messages will provide a useful standard format. Specific EDIFACT messages have been defined for payment order, account information etc. All of the required security services will be part of the ISO standard 9735-5: confidentiality, integrity, authentication and non-repudiation.

EDIFACT messages can be transferred with, e.g., *SEMPER* containers [SEMP96]. The payment gateway is needed to enable the user's application to exchange messages with his bank:

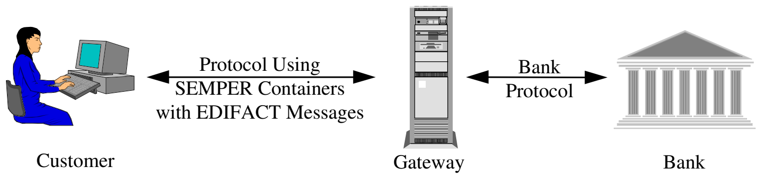

Figure 8: Payment gateway maps between protocols using *SEMPER* containers with EDIFACT messages and bank protocols.

3. Architecture of the Payment Gateway

3.1 Protocol Mapping

The outputs the payment gateway creates in direction to the banking network are themselves inputs to the banking side, e.g., a payment request is formatted exactly in the way the bank needs it for internal processing. The banking side itself will process this message internally, according to the corresponding bank protocol (and may, e.g., send back a response).

Towards the Internet side, the payment gateway will represent the bank, talking the 'Internet language'. Towards the banking side, the payment gateway will represent the customer, or merchant, respectively, talking the 'banking language'. Thus the payment gateway ***mediates*** between any two players (between the customer and the bank, or between the merchant and the bank) - without being a player itself.

The transaction protocols used at the Internet side might be very different from those used at the banking side. Thus, a generic payment gateway has not only to translate messages but also to ***map transaction protocols***. For instance a SET capture request can include a batch of capture tokens that need to be mapped to a sequence of capture requests over the bank network. Another example that the messages may not be mapped one-to-one: a request on the bank network is mapped to a request on the Internet and - if no response is received within a certain time - one or more retries.

Please note that often when we speak of ‘protocol’ here and in the following, we always mean the local protocol (transaction) part of one player (and avoid the term ‘part’) which gives the rules to take part in the complete protocol. It should be clear from the context what is meant.

Not all Internet protocols can be mapped into each bank protocol. There is a separate mapping for the four Internet protocol classes:

- credit-like
- cash-like
- electronic check
- electronic fund transfer.

As an example Figure 9 illustrates the mapping for credit-like protocols. On the Internet side we have, e.g., the protocols SET and iKP. These protocols are handled by modules implementing the corresponding Internet protocol. They are responsible for all cryptographic operations, e.g., checking of signatures and decryption in case of an incoming authorisation request. The Internet protocol is then mapped to a generic credit-like protocol (i.e. part of the intermediate protocol, see Section 3.2). For credit-like mapping the payment module forwards a credit-like transaction from the Internet side to a credit-like transaction on the bank side (some transactions on the Internet side may require several transactions on the bank side and vice versa). Technically a credit-like protocol could be mapped to a protocol of another type, e.g., a cash-like protocol. The credit-like module on the bank side decides according to the BIN (bank information number, it is part of the credit card number) which credit card organisation has to receive the authorisation request and uses the bank protocol suitable for the acquirer. The authorisation response from the bank network flows in the opposite direction through the gateway.

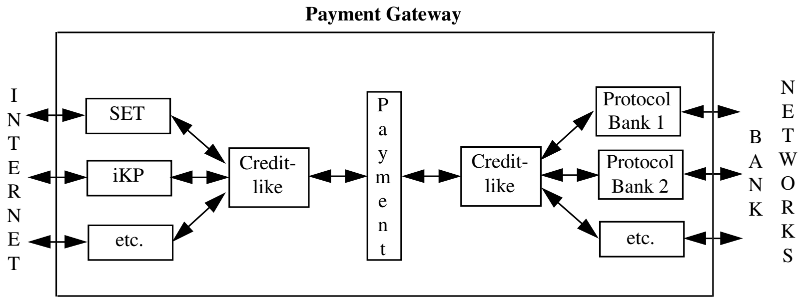

Figure 9: Mapping credit-like protocols.

3.2 Intermediate Protocols

Assume for a type of payment instruments like credit-card payments that there are m payment protocols on the Internet and n protocols on the financial network. A straightforward complete mapping - which we do not intend to use - would require m x n protocol mappings. Moreover, adding one Internet payment system requires to add n mappings! Similarly, adding one bank protocol requires to add n mappings.

In order to avoid all these protocol mappings in the gateway, we suggest to define an intermediate protocol and mappings between this protocol and each banking and Internet protocol. This will reduce the number of protocol mappings to $m + n$ (Figure 10), which is less than or equal to m x n if both m and n are at least 2 (i.e., more than one protocol on both sides), and for larger values of m and n the difference grows considerably. Adding one Internet payment system or one bank protocol requires only one additional protocol mapping!

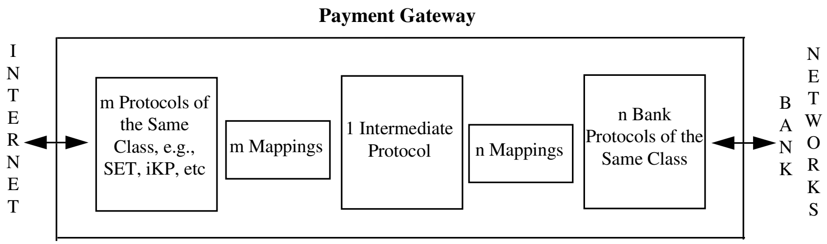

Figure 10: The intermediate protocol allows $n + m$ mappings instead of n x m mappings.

For a payment gateway that supports credit-like, cash-like, e-check and EFT protocols there are, in principle, different intermediate protocols. However, overlapping can be used to save overhead and to enable using different types (or part of them) in the same transaction. In the remainder of this report, we consider one intermediate protocol for each type of payment system, for the sake of simplicity.

These intermediate protocols will be representatives of types of payment systems, i.e., one for credit-like, one for cash-like, one for e-check and one for EFT.

So the intermediate protocol is expected to be more complex than any Internet or banking protocol, and each protocol mapped into the intermediate protocol is a kind of subprotocol of the intermediate protocol, not necessarily using all states or all inputs/outputs. The point is that the intermediate protocol builds a kind of 'bridge' so that also rather different protocols at each side can be mapped to each other.

3.3 Architecture

In Figure 11, the components of the payment gateway are depicted and will be explained here shortly. We will concentrate on the mapping-related functionality of the payment service block. The other blocks which will be used in the same way as for buyer’s and merchant’s *SEMPER* software [SEMP96]:

- The **business application** could be used to implement business applications involving the bank as a business partner, e.g., for selling bank services or shares. This might be the case if the bank prefers to use the payment gateway to put new business applications on, using the gateway for mapping to given bank protocols as far as necessary, rather than to implement new applications at the bank side. The business application will not be existent if the gateway's role is confined to mediate the protocols between the Internet and the bank network - the protocol mapping will take place completely in the lower layers.
- The **transfer manager** parses the messages and gives them to the payment service block or statement service block. Vice versa, messages are assembled, e.g., containers are sent over the Internet. The 'upper' methods *receiveContainer()* and *sendContainer()* will be present here, too. Apart from the messages coming from the Internet side (via the transfer manager), the transfer manager takes also messages from the bank interface.
- The **payment service block** handles each single payment transaction, thereby performing all necessary mappings between Internet and bank protocols. The connection of the gateway to several banks requires that each bank is represented by a separate bank protocol in the payment service block (which can be achieved, e.g., by using the same protocol but different parameters). The mapping related operations will be explained in more detail in Section 3.4.
- The **statement service block** processes ‘statement nodes’, like any (signed) information not necessarily related to a payment.
- The **certificate service block** checks the user’s certificates or assembles the bank’s certificates.

We can identify the following common **supporting services** used by several components of the architecture:

- The **communication** services support communication between entities on different devices. Input and output to the Internet are handled via the **Internet communication service block**. The architecture can support any other network than the Internet, e.g., a broadband ATM-based network. An appropriate communication service block is required for this purpose. Input and output to the bank network are handled via the **bank communication service block**.

- The **cryptographic service block** provides cryptographic primitives like message encryption and decryption, hash functions, message authentication codes, digital signatures, and key generation. For performance and security reasons special hardware may be required (see below: tamper resistant hardware).
- The **archive service block** supports persistent and secure local storage of data. For instance, the archive may be used to store received and sent digitally signed messages, certificates and cryptographic keys (e.g., public/private key pair of acquirer).

  The archive will also be used for log and local audit. All incoming and outgoing transactions can be logged. Additional information may be required for recovery purposes. Optionally a **CD-WROM** may be used as an additional persistent memory module for the archive manager. On this CD all the information of incoming transactions and outgoing converted transactions could continuously be recorded and kept for evidence (this also requires support for the evaluation of the recorded data).
- Optionally **tamper resistant** hardware could be used to protect the acquirer keys. This makes sense only if also the computations with these secret keys (crypto block) are done by this tamper resistant device, i.e., the secret keys must never leave the device.

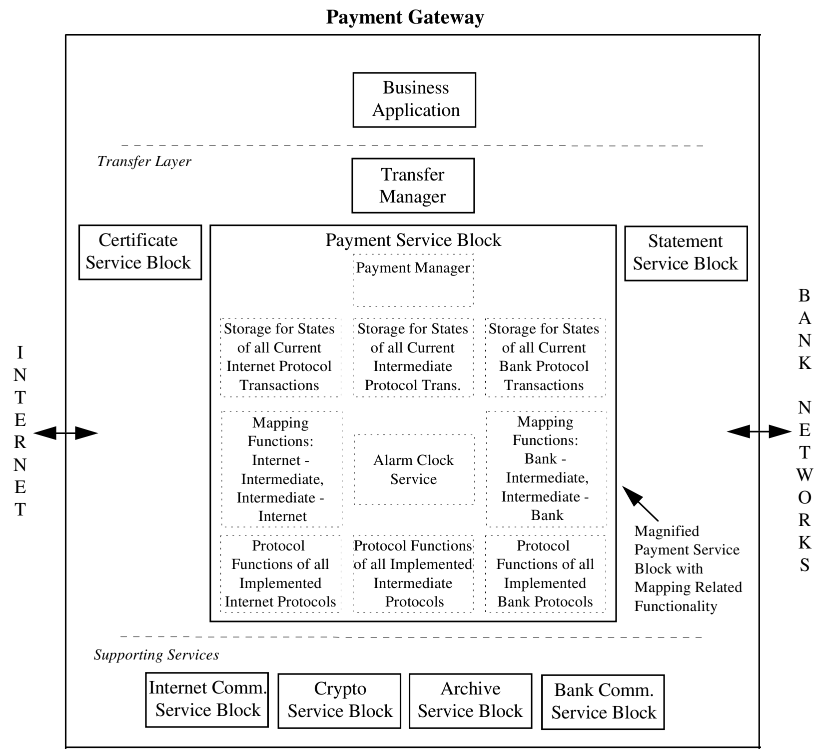

Figure 11: Architecture of payment gateway.

3.4 The Payment Service Block

The payment service block will enable the payment gateway to

- behave as one or more Internet players (i.e., payer, payee) towards the bank to enable them to perform the bank protocol with the bank and
- behave as the bank towards the Internet players (i.e., payer, payee) to enable the bank to perform the Internet protocol.

In this section we will propose a way to perform the necessary mapping and to support at the same time fault tolerance and extendability.

3.4.1 Alarm Clock Service and Time-outs

Time-outs are defined in order to stop waiting for expected messages that might arrive too late or never:

If an expected message (e.g., a response) does not arrive before a time-out expires, the event 'time-out alarm' will be processed instead of that message. In the latter case the protocol may define, e.g., that the request should be retried, or that the protocol will be finished unsuccessfully, or an error message will be sent to a player or component.

The time-out handling can be realised by an additional component acting as alarm clock (already given by many operation systems).

Whenever a process is going to enter a state where it has to wait for some message, it requests a 'time-out alarm' and waits for the event that arrives first: the expected message or the 'time-out alarm', respectively.

When a message arrives 'at the same time' as its time-out, one of them will be taken as input and the other will be ignored.

3.4.2 Other Payment Gateway Internal Services

Apart from the alarm clock service, there may be other services needed, for instance a database which stores all received coins to prevent double-spending. In this case, the Internet protocol within the payment gateway gives each received coin to this database and waits for the answer. The database will check if the same coin has been received before and answer, e.g., 'ok' or 'double-spent error', and store the coin in case is has not been stored before. The Internet protocol within the payment gateway receives the answer of the database component. In case of 'ok' the payment can be mapped via the intermediate protocol to the bank protocol. In case of 'double-spent error', the Internet protocol at the payment gateway refuses the coin and sends back a message to the peer entity at the Internet.

Other internal services can be a database storing certificates, and crypto components which check certificates.

3.4.3 Transaction-ID

Transaction-IDs carried within messages are needed as the payment gateway will be handling hundreds or thousands of transactions at the same time, and therefore it is necessary to know which message belongs to which transaction.

Each transaction gets a transaction-ID when it is initiated. The initiating player issues this unique transaction-ID. Any retries of the initiation will have the same transaction-ID. For the gateway, the initiating player will in most cases (if not always) be the payer or payee on the Internet rather than the bank.

3.4.4 Protocol and Mapping

In this section we will focus on the relationship between the Internet and bank protocols and their mapping.

3.4.4.1 Protocol Notation

We will model the protocol behaviour for each single player participating in a protocol (e.g., a payment protocol) as an (extended) **finite state machine, which is defined by a number of states** (q) **and state transitions.** We use 't' to mean logically the same time (period) from one event to the next but not necessarily also physically the same as there will be delays caused due to computing. The time point 't+1' is the logical time (period) after the next event.

The state at time t is denoted by q^t. Depending on the received input i^t, the machine will make a transition to state q^{t+1} and produce the output ($\underline{o}^t$). This means, each protocol behaviour of any player for a certain protocol can be described by giving the corresponding function f_q for the state transitions (1) and the function f_o for the outputs (2), both depending on the current state and the inputs:

(1) $q^{t+1} = f_q(q^t, i^t)$

(2) $\underline{o}^t = f_o(q^t, i^t)$

$\underline{o}$ is a vector as the number of outputs for one state transition is not restricted to one. Unlike an input which is always equivalent to one event, an output (vector) can cause more than one events.

As we give protocols as finite state machine, the role of inputs i^t and outputs $\underline{o}^t$ are the following:

- ***input*** i^t is a received message (e.g., an 'Internet' message passed by the payment manager), a received time-out alarm, or another event (we will later come back to 'mapped inputs'),
- ***output*** $\underline{o}^t$ are messages to be sent (e.g., 'bank' messages passed to the payment manager) and/or messages requesting time-out alarms. $\underline{o}^t$ also can be empty, unlike i^t.

The complete set of equations for one protocol (i.e. the protocol behaviour for one player) could alternatively be illustrated by a state transition diagram or a table; all these illustrations are equivalent to each other:

- A ***state transition table*** contains a row for each possible q^t, and a column for each possible input i^t.

All fields "q^t, i^t" where input i^t (of that column) is actually one of the potentially expected inputs for the corresponding state q^t (of that row) are filled with the subsequent state and output, i.e., contain the entry "q^{t+1}, $\underline{o}^t$".

All other fields are empty ("-").

- A ***state transition diagram*** contains all possible states q^t, each drawn as a separate button.

  For each state q^t, each possible input (expected for this state) defines an arrow leading from q^t to the subsequent state q^{t+1}, caused by input i^t. The inscription of this arrow is "i^t, $\underline{o}^t$".

  If different inputs for q^t lead to the same state q^{t+1}, only one arrow is needed here with several inscriptions, for the sake of clearness. (Arrows may also lead to the source state itself.)

The ***interaction of protocols*** is usually done by taking certain outputs of one protocol (i.e. one state transition diagram) as inputs for certain counterpart protocols (with other state transition diagrams).

3.4.4.2 Neighboured Protocols

Usually, protocols are designed, as described in Section 3.4.4.1, to be run commonly by certain parties to perform some transaction. For example, a payment protocol may involve a bank, a payer and a payee to perform a payment from the payer to the payee.

In this case, the protocol behaviour of each party, given as equations as described in Section 3.4.4.1, uses messages as inputs and outputs:

- The Internet protocol in the payment gateway will send and receive messages to and from the Internet players involved.
- The bank protocol in the payment gateway will send and receive messages to and from the bank.

For simplicity, we restrict consideration to one transaction to be performed by the payment gateway. For this transaction, and the corresponding transaction-ID, it should be clear (e.g., by the corresponding parameters contained in the messages) which one of the *m* Internet protocols, and which one of the *n* bank protocols is currently performed.

In the following we will consider the mapping between an Internet protocol and a bank protocol. Thus we assume that we have:

- Internet protocol
- intermediate protocol
- bank protocol.

Since the mapping is the main subject of interaction between these 3 protocols, and since they are executed by the **same component**, i.e., the payment gateway, and as they may use resources in common (e.g., processors, disk), they are referred to as **'neighboured'** to each other here.

The 'neighboured protocols' also interact with each other in a certain sense, but **unlike** in the way as two parties do in the same protocol:

The 'Internet protocol' within the payment gateway performs an Internet protocol (via payment manager and Internet communication service block) with the counterpart players payer and/or payee, representing the bank towards the payer and payee, respectively.

Correspondingly, the 'bank protocol' within the payment gateway communicates (via payment manager and bank communication service block) with the bank, representing all involved Internet players (e.g., payer and payee) towards the bank.

3.4.4.3 Mapping Between Neighboured Protocols

To describe the mapping of protocol execution among the neighboured protocols, we first need the protocol equations for the three protocols contained in the payment gateway. Assume protocol x is described by a pair of functions (f_{q_x}, f_{o_x}), where x is "int", "im" or "ba" for the Internet, the intermediate and the bank protocol, respectively. Thus the Internet protocols is described by:

(1_{int}) $\quad q_{int}^{t+1} = f_{q\_int}(q_{int}^{t}, i_{int}^{t})$

(2_{int}) $\quad \underline{o}_{int}^{t} = f_{o\_int}(q_{int}^{t}, i_{int}^{t})$.

The intermediate protocol is described by:

(1_{im}) $\quad q_{im}^{t+1} = f_{q\_im}(q_{im}^{t}, i_{im}^{t})$

(2_{im}) $\quad \underline{o}_{im}^{t} = f_{o\_im}(q_{im}^{t}, i_{im}^{t})$.

and the bank protocol by:

(1_{ba}) $\quad q_{ba}^{t+1} = f_{q\_ba}(q_{ba}^{t}, i_{ba}^{t})$

(2_{ba}) $\quad \underline{o}_{ba}^{t} = f_{o\_ba}(q_{ba}^{t}, i_{ba}^{t})$.

We assume that all 3 protocols are given in this way. We are considering now the way they will co-operate. It is obvious that they will have to pass any information to each other since information received from the Internet will have to be forwarded to the bank, and vice versa, aided by the payment service block.

It is ***sufficient to map those inputs*** which are needed by neighboured protocols. The outputs need not to be mapped because of the fact that all outputs in one state transition depend directly on the input and on all information available in the current state (e.g. the information received by all previous inputs, as well as the 'knowledge' of all previous states).

For the same reason as for the inputs, we do not need separate equations for the mapping of states - this will be performed implicitly, caused by the mapping of inputs.

When designing these three protocols, the protocol designer has to take into consideration in which way the protocol states have to be mapped on each other. He has to design the protocols for the gateway in such a way that mapping inputs is sufficient to map the states implicitly during the protocol execution.

In the same way, the protocol designer has to give the concrete mapping equations to map the inputs during the protocol execution where appropriate.

The mapping of a certain input i_x^t of protocol **x** to a corresponding input i_y^t of protocol **y** (where x and y can be **int** and **im**, or **im** and **ba** etc.) can be specified by giving a function $f_{imap\_x\_y}$ in the following way.

(3_{x_y}) $\quad i_y^{t} = f_{imap\_x\_y}(q_y^{t}, i_x^{t})$, $\quad$ for some i_y^t.

This means, the mapping of an input i_x^t from protocol **x** to protocol **y** does not only depend on the input itself but also on the current state q_y^t of the destination protocol **y**.

Thereby the equation (3_{x_y}) need not contain q_x^t as an argument for $f_{imap\_x\_y}$ in our approach.

The reason is that, theoretically, you could map ***all*** inputs (rather than just *some* inputs) from protocol **x** to protocol **y** to have the maximum information of each state q_x^t at protocol **y**.

We suppose the protocol designer to reduce this exhaustive mapping as far as possible. (We can argue in this way using the exhaustive mapping mentioned as we know that in our approach, 'neighboured' protocols do not map their inputs in circles but in a sequence only.)

3.4.4.4 Special Cases

There will be inputs where q_y^t is irrelevant in equation (3_{x_y}), leading to a pure 'message mapping' of i_x^t to i_y^t, without regarding q_y^t.

There will also be inputs where $f_{imap\_x\_y}$ is only the identical mapping for some q_y^t, i.e., where i_y^t will just result in i_x^t.

3.4.4.5 Resulting Inputs for Neighboured Protocols

Inputs for the three 'neighboured protocols'**, int, im** and **ba,** can result from several events. The first input available will be processed. For each state, the considered protocol has to check if one of the possibly expected inputs is available:

- Is there a 'real' input, i.e. a message received (via payment manager) from an Internet player or the bank, available which is expected for my current state?
- Is there any input available from my neighbours the mapping of which is an expected input for my current state?

  (Since this check and the mapping itself does not belong to the protocol itself, we extract this functionality into a separate function get_map, Section 3.4.6)
- Is there a local event, e.g., a time-out alarm, or result of a computation, available, which is expected in my current state?

We will give these possibilities for each of the 'neighboured protocols' **int, im** and **ba**, separately:

3.4.4.5.1 Inputs for the Internet Protocol

The Internet protocol **int** performed by the payment gateway gets its input i_{int}^t in one of the following ways:

i_{int}^t is

- a message received from an Internet player, via the payment manager, or
- a mapped input

  $f_{imap\_im\_int}(q_{int}^t, i_{im}^t)$, if defined,

  from the neighboured protocol **im**, or

- a local event for protocol **int**, e.g., a time-out alarm.

3.4.4.5.2 Inputs for the Intermediate Protocol

The intermediate protocol **im** performed by the payment gateway gets its input i_{im}^t in one of the following ways:

i_{im}^t is

- a mapped input

  $f_{imap\_int\_im}(q_{im}^t, i_{int}^t)$, if defined,

  from the neighboured protocol **int**, or
- a mapped input

  $f_{imap\_ba\_im}(q_{im}^t, i_{ba}^t)$, if defined,

  from the neighboured protocol **ba**, or
- a local event for protocol **im**, e.g., a time-out alarm.

3.4.4.5.3 Inputs for the Bank Protocol

The bank protocol **ba** performed by the payment gateway gets its input i_{ba}^t in one of the following ways:

i_{ba}^t is

- a message received from the bank, via the payment manager, or
- a mapped input

  $f_{imap\_im\_ba}(q_{ba}^t, i_{im}^t)$, if defined,

  from the neighboured protocol **im**, or
- a local event for protocol **ba**, e.g., a time-out alarm.

3.4.4.5.4 Resulting Character of the Intermediate Protocol

In a straightforward way, the intermediate protocol could be designed to reflect each input of the Internet protocol, as well as the mapping of each input of the bank protocol, thereby including also each possible time-out and each other possible event.

Though the intermediate protocol should be designed in a less exhaustive way, it is supposed to keep this character.

3.4.4.5.5 General Remarks on Protocol Mapping

From the preceding sections, it is obvious that inputs are not necessarily mapped one-to-one from one protocol to its neighboured protocol.

An input to one protocol may cause a sequence of more than one state transitions at a neighboured protocol. An example for this situation is that the Internet protocol receives a SET message including a batch of capture tokens which is mapped into a first capture request, and after receiving the response of the bank (this is another event!), additionally into a second capture request etc.

Another example is that one single message of the bank could result in a complete protocol exchanging several messages between the gateway and the Internet players involved, or, in a similar way vice versa, that a complete protocol results in only one message finally being sent to the bank.

The reason for the existence of the latter examples is that the payment gateway has to perform all transactions to authenticate the identity of the Internet player - the bank protocol need not deal with such matters of trust.

Generally, one banking transaction may serve more than one Internet transaction, and vice versa.

Furthermore, an Internet message might not be mapped 'through' the payment gateway at all: it could result in an intermediate message which might result in an Internet message back again, e.g., due to insufficient information or errors.

So the direction of mapping is not necessarily always 'straight through' the payment gateway.

3.4.4.6 No Output Mapping if all Inputs are Accessible

In our approach, we use the fact that the outputs need not to be mapped 'through' the payment service block if all inputs of the neighboured protocols can be accessed. Apart from 'internal' outputs, e.g., to request a time-out notification, it is sufficient then that a bank or Internet related output is created just before it is passed to the payment manager.

Outputs are

- a message sent to the bank (via the payment manager)
- a message sent to some Internet player (via the payment manager)
- a local request for a time-out, or the request for a computation to an internal component, or some other request for information to any payment gateway-local component.

3.4.4.7 Output Mapping if not all Inputs are Accessible

We would like to give some remarks on the integration of payment gateway tools for certain payment systems (e.g., for SET, or Ecash®) which may prevent the way of pure mapping of inputs - since some inputs may not be accessible then and thus a pure mapping of inputs may not be possible. If we do not use such tools at all, we can assume that all inputs can be made accessible.

- If we know the specification of the tool (i.e., can deduce an equivalent state machine) and can access all inputs including the internal ones like time-out notifications, then we can use this tool and the mapping can be performed as proposed in earlier sections. We could even map the protocol into our own intermediate protocol to enable mapping to other bank (or Internet) protocols.
- If not all inputs are accessible and we know the specification of the tool, than we can spend some additional functionality to deduce these inputs from the outputs produced by this tool and those inputs which are available. If this is possible in all cases, then the same situation is achieved as in the upper point.

  If this is not possible in all cases, however, then an exact mapping is not possible. Either we can spend some functionality to deal with certain exception cases separately, or we are forced to make a new design and implementation based on the known specification, resulting in the same situation as in the upper point.
- If we do not know the specification of the tool (i.e., cannot deduce some corresponding equivalent state machines), it is principally difficult and may even be impossible to enable a general mapping to other protocols by using the intermediate protocol.

Further work will give more exact guidelines how to integrate given tools depending on the given interfaces, and in which ways they have to be used for mapping.

Another point will be investigated in more detail. We mention it here since this idea came up and is worth to be considered because it can save much effort:

It could theoretically be possible that for some of the Internet protocols and the bank protocols it is completely sufficient to map their *outputs* instead of the inputs. In this case, we would not need the mapping of inputs but would do it by taking the outputs for the mapping. Furthermore, this could simplify the intermediate protocol as the latter would not have to reflect all corresponding states of both neighboured protocols. As outputs are sufficient for mapping here, the integration of payment system tools would be easier.

However, we are pessimistic that this simplified approach would work. The reason is that outputs do not necessarily reflect the states which caused them, and important mapping information may not be available this way. In the remainder of this report, we therefore ignore this approach - but keep it in mind to be investigated by further work.

3.4.5 Transaction State Storage and Input Queues

Each new transaction-ID requires an allocation to store the current states for each of the three protocols (Internet, intermediate, bank) with a reference to this transaction-ID. Three input queues for each transaction-ID are opened, one for the Internet protocol, one for the intermediate protocol, and one for the bank protocol (see Figure 12). As soon as the transaction is finished, the states can be deallocated again.

To employ separate queues for each transaction-ID has the advantage that different priorities for different transactions can be assigned and handled on a higher level, independently of the transaction input queues.

3.4.6 Getting Input Events by Function Get_map

Each input event (see Section 3.4.4.5) is stored into the corresponding queue for the corresponding transaction-ID.

That an input belongs to a specific queue can be recognised, e.g., by the transaction-ID carried, as well as the destination protocol (i.e., one of the neighboured protocols). So incoming inputs (via the payment manager) as well as local events (e.g., time-outs) will be stored into the corresponding queue.

Slightly different handling is needed for protocol **y** to get the 'mapped' inputs $i_y{}'$:

(3$_{x_y}$) $i_y^{\,t} = f_{imap\_x\_y}(q_y^{\,t}, i_x^{\,t})$, for some $i_y^{\,t}$.

Protocol **y** cannot expect to get the mapped input from neighbour protocol **x** - this would extend the protocol behaviour of protocol **x**.

In a similar way, it would extend the behaviour of protocol **y** to check the queues for the neighbour protocols for new inputs available to be mapped.

So we introduce here a new component to perform the function **get_map_y** which has the task

- to know the current state $q_y^{\,t}$ of protocol y. From this also follows the 'knowledge' which inputs are potentially expected for this state and where they may come from (e.g. mapped from protocol **x**).
- to look if input $i_x^{\,t}$ is already available in the queue for protocol **x** (for this transaction-ID), and, if so,
- to compute $i_y^{\,t} = f_{imap\_x\_y}(q_y^{\,t}, i_x^{\,t})$.

This implies that all inputs of the three neighboured protocols must remain available for all get_map components.

As only one Internet protocol and only one bank protocol process the same transaction-ID, an input event can be marked by all neighbours (more precisely, by their get_map component) which have already processed it - so the event can be deleted if it has been processed by all corresponding neighbour protocols which needed it (there is one neighbour protocol for the Internet or bank protocol and there are two neighbour protocols for the intermediate protocol).

Another easier way is to store all inputs until the end of the transaction, each marked by all components which processed them. This is more helpful also for fault tolerance mechanisms.

3.4.7 Processing of Events

One or more processes are needed to process the state transitions for one transaction performed by the three neighboured protocols depending on available inputs, and to compute and pass the outputs. Also the mapping **get_map** has to be performed for each of the neighboured protocols.

Since the current state is stored for each of the neighboured protocols, respectively, and since the inputs are stored in queues, it is **not** necessary to employ exactly one process to perform exactly one complete transaction. Nearly at any time, the process can stop, and another process may (i.e. should) continue later. This allows the allocation of processes in a most flexible and scaleable way.

The most reasonable way is if the process stops only as soon as no more inputs are available. Periodically any (other) process should check then if the transaction can be continued. All this can be designed and handled separately from the processing of events, thus enabling a most modular concept.

Even more than one process could work independently on different events (in different neighboured protocols) for the same transaction. There are solutions without writable shared data.

The flow of information within the payment service block is given by Figure 12.

3.4.8 Additional Payment Systems

If new payment systems have to be added, only the new protocol functions and the corresponding mapping functions (to be mapped to or from the intermediate protocols) have to be supplied.

The aim here is to design the intermediate protocol in such a way that any additional Internet protocol or bank protocol will not cause any modifications of the intermediate protocol - apart from its additional function get_map (which is strictly spoken not part of the intermediate protocol).

3.4.9 Scalability

The rate of transactions per time can be increased by increasing the number of processes running at the same time. Multiprocessor systems or parallel computers are indispensable for a high amount of transactions in practical use. Along the number of processes, also the memory and storage place has to be adapted correspondingly. To save computation time, special hardware should be used for cryptographic functions.

Our architectural approach should support a high increase of transactions, e.g., at the change from a trial to a regular exhaustive use. The states for each transaction are stored in such a way that any processor may continue at any time (best within reasonable time-outs), so that the scheduling and allocation of processes can be performed in a highly flexible way.

The usual problems of adding disk space, memory and processors are independent of and will not be increased by additional gateway-specific problems.

3.4.10 Fault Tolerance

Apart from the fact that all protocols have to be designed to be fault tolerant, some payment gateway specific points will be considered here.

By redundant storage of the states, a fault-tolerant solution can be given: The processing of inputs should be organised such that each input must not be deleted from the queue before it has been processed completely including the resulting state transitions. The 'old' states are stored temporarily.

Helpful is the approach to store the inputs until the end of the transaction. Additionally to the inputs, the information should be stored which neighboured protocols have already processed or mapped the corresponding inputs.

After a system crash, the pending transaction can be recovered using the stored 'old' state - which corresponds to the current state if no input has been processed in the meanwhile, or to the 'old' state otherwise, i.e., the state before the previous input has started to be processed. The current inputs (which were to be processed) should still be existent, and it should be possible to check (aided by the stored state, the knowledge of the protocol and id's within the input, or aided by marks) if they have already been processed or not.

The problem left to be solved is the decision (which may be lost) if the output has already been given to the banking side (may be easy to check?) or to the Internet side - it may not be possible to decide the latter (was the crash just before or just after the sending?). In doubt, the same message (with the same transaction-ID, of course) could be simply sent 'again'. If it was the second one of the same message, the recipient will handle it as 'unexpected' and ignore it.

The problem of messages arriving more than once has anyway to be solved during designing the Internet protocol, or the bank protocol, respectively. Independently of the payment gateway, any protocol should be able to deal with this (as this may happen for several reasons). In the same way, all protocols should be able to deal with lost messages, for whatever reasons. Sometimes, it may not be possible to complete the transaction successfully, but at least a consistent view among all players involved should be achieved on the fact that something has gone wrong and (part of) the transaction should be repeated. Double-spending of the same electronic cheques, for instance, are prevented by other means, having been taken into account (by all payment protocols which currently exist) when the payment protocols were designed. In the same way, all business transactions will deal with lost or duplicated messages (otherwise they should better be re-designed).

If redundant storage is used for fault tolerant purposes, measurements to keep consistency have to be taken additionally.

3.4.11 Flow Diagram

The flow of events and the actions to be taken in the payment service block can be depicted with a flow diagram (Figure 12).

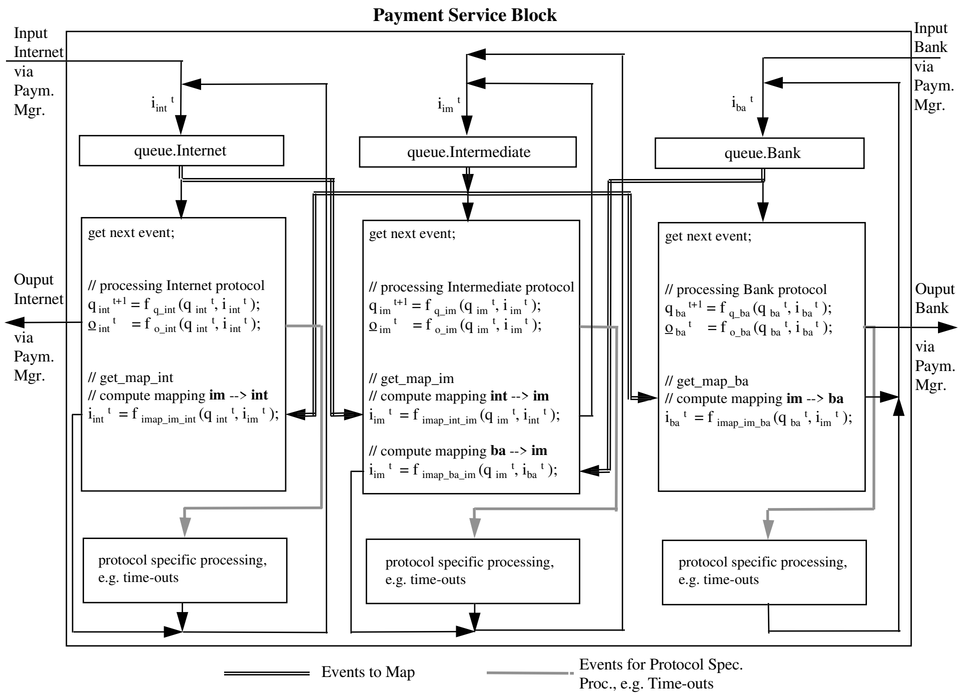

Figure 12: Flow diagram of payment service block.

3.4.12 Advantages of this Approach

The advantages of the described approach are:

- Generic architecture providing interoperability between various Internet payment protocols and bank protocols.
- Full support of the *SEMPER* architecture which is independent of specific payment protocols and - at the same time - is able to include all reasonable payment protocols.
- Reduction of the number of protocol mappings to a minimum using an intermediate protocol.
- Extendability and flexibility: The effort to add later new protocols on the Internet or bank side is reduced to a minimum. Similarly, the architecture supports ease of change, for example to modify a bank protocol.
- Scalability and throughput: The architecture supports scalability independently of the processing of protocols. This is achieved by a highly modular concept offering a most flexible way for the scheduling of processes.

4. Connection to the Internet

The payment gateway is connected to the highly untrusted Internet. An attacker could brake into the gateway through the Internet connection and get access to the data on the gateway and the bank networks. The data on the gateway include the acquirer's private keys!

Therefore the gateway must be protected with the utmost care including a very limited Internet connection using advanced firewall technology [CZ95]. A firewall (see Figure 13) needs to 'filter' all incoming messages from the Internet so that the gateway receives only and exclusively the containers related to payment transactions:

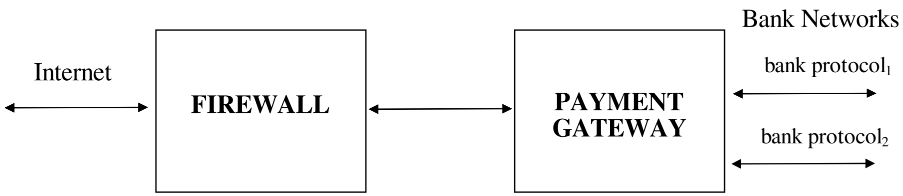

Figure 13: A firewall needs to protect the payment gateway from attackers on the Internet.

5. Conclusions

5.1 Summary

As a smooth completion of the *SEMPER* architecture, a generic system architecture for a payment gateway to connect closed financial networks to open networks has been given. It supports all relevant payment systems and transactions identified involving financial institutions. It also enables the addition of further services later.

The proposed payment gateway enables a financial institution to offer all Internet payment instruments by just providing an interface to their bank network. Similarly, any new Internet payment instrument can easily be made available to all banks by just providing an adapter for the corresponding Internet payment instrument.

Due to a highly modular concept including 'intermediate protocols', the proposed architecture reduces the number of protocol mappings to a minimum and enables realisation, extendability and scalability in a most efficient way.

5.2 The Next Steps

To take full advantage of the described architecture, the design of the intermediate protocol(s) is a critical task. It requires a careful analysis of all relevant payment systems and transactions.

The intermediate protocol has at least to realise a 'superset' of:

- the protocols to be supported on the Internet and bank side
- those protocols to be supported in the future (e.g., currently not implemented protocols and/or new emerging protocols)
- the protocol modifications that might be expected in the future
- all reasonable possibilities to perform exception handling, additionally to those involved in existing protocols.

The intermediate protocol should remain stable even if Internet or bank protocols are modified, extended or added. It would be acceptable to extend the intermediate protocol in a way that does not affect any mapping functions already implemented.

A first approximation of an intermediate protocol (for each payment type) will be designed and validated using existing Internet and bank protocols. This approach will be refined subsequently. The resulting intermediate protocol should be able to build a reasonable basis for the mapping of the existing and future protocols.

6. References

[BGHH95] *M. Bellare, J. A. Garay, R. Hauser, A. Herzberg, H. Krawczyk, M. Steiner, G. Tsudik, M. Waidner:* **iKP - A Family of Secure Electronic Payment Protocols;** *Usenix 1995.*

[CZ95] *D. B. Chapman, Elizabeth D. Zwicky:* **Building Internet Firewalls;** *O´Reilly Associates Inc., 1995.*

[DIGI96] *DigiCash***:** **ecash;** *<http://www.digicash.com/>*

[FSTC95] *Financial Services Technology Consortium***:** **FSTC Electronic Check Project;** *<http://www.fstc.org/>*

[ISO93] *International Organization for Standardization:* **Financial Transaction Card Originated Messages - Interchange Message Specifications;** *ISO 8583, Second Edition; 1993.*

[JW95] *P. Janson, M. Waidner:* ***Electronic Payment over Open Networks;*** *SI Informatik/Informatique 3 (1995) 10-15.*

[MV96] *Mastercard, Visa:* **Secure Electronic Transactions;** *Draft, Aug 1, 1996; <http:///www.mastercard.com/set/set.htm>.*

[PAW96] *J. Abad Peiro, N. Asokan and M. Waidner:* **Payment Manager - Overview**. *Public SEMPER report 212ZR054. March 21, 1996; <http://www.semper.org/info/>.*

[SEMP96] *SEMPER Consortium:* **Basic Services, Architecture and Design;** *Public Project Deliverable D03; Aug 31, 1996; <http://www.semper.org/info/>.*

[STIN95] *Douglas R. Stinson:* **Cryptography: Theory and Practice;** *CRC Press, 1995.*

7. Appendix: Glossary

Acquiring bank
The bank used by the recipient of electronic money.

Audit trail
Logging information which is the basis for an audit of the use of a service or system. The audit trail may contain non-repudiation tokens to resolve disputes.

Authentication
A proof of identity. This proof can be related to a message being sent meaning that a proof of origin is sent along (see also digital signatures and message authentication codes).

Authorisation
The process of determining the rights associated with a particular principal or role, e.g. rights to access data or services on a computer system.

Business session
A business session is a sequence of commerce layer primitives that operate on the same context, including all case decisions, e.g., for error handling. The primitives could support, e.g., offer, order, invoice and payment.

Business sub-session
A meaningful part of a business session which could itself be called a business session.

Certificate
A certificate is a digitally signed statement about a person. *SEMPER* distinguishes three types of certificates: key certificates linking a public key to a person, attribute certificates linking a particular property to a person and hybrid certificates linking both a key and an attribute to a person.

Conditional access
Access to a service, which is restricted to entities having certain properties or rights.

Confidentiality
Protecting a message against eavesdropping such that it is only meaningful to the intended set of entities.

Container
A data structure used in *SEMPER* to transfer or exchange information and payments. It is a data type structured in the form of a tree. The leaves are nodes specifying or containing (protected) information or payments. Internal nodes contain security attributes.

Context
The context is the local state of an entity. The context evolves from the configuration and preferences set by the user and the results of negotiating parameters with other entities.

Credentials
The personal information provided by a person or institution when registering. (In the literature, the term credential is also sometimes used instead of hybrid certificate).

Customer
The role played by someone wanting to buy something at a merchant.

Decision procedure
A procedure which is used to solve disputes between participants - usually based on audit trails. The decision procedure defines which messages serve as evidence of what and how they are verified.

Digital signature
A digital signature is an electronic counterpart of hand-written signatures. It is verified against a public key, and if the signature system is secure, this serves as a proof that the signed message originates from someone knowing the corresponding secret key

Dispute handler
See Exception handler.

Electronic money
Electronic money is the electronic counterpart to the conventional money.

Entity
Any active element in the *SEMPER* architecture is called an entity. Thus an entity could be a service block, a manager, a module or a collection of these.

Exception handler
An exception handler has to resolve an exception raised by a party. In a first approach the exception handling bases on the assumption that all parties are honest. If this optimistic approach fails, the parties are in a dispute and a more pessimistic approach must be used, usually involving arbiters and finally courts. Exception handling requires all parties to keep sufficient audit trails. For real disputes, the audit trail has to contain evidence, e.g. non-repudiation tokens.

Exchange
An exchange is a protocol whereby a number of parties can exchange business items (payment, information, vouchers, etc.) . In the simplest case these are two transfers.

Fair exchange
A special type of exchange in which each party beforehand specifies what he wants to send and receive. The exchange is fair if the participant is guaranteed to receive an item fulfilling this specification if he sends the promised item.

Hash functions
A function used to compute a fixed length digest of any bit string. Unless stated otherwise a hash function is always assumed to be *collision-resistant*, meaning that it is not feasible in practice to find two different bit string with the same digest.

Identification
A process whereby an entity proves its identity.

Integrity protection
The protection of information against changes by intruders (tampering)

Issuing bank
The bank of the payer in an electronic payment systems.

Non-repudiation tokens
A non-repudiation token is a proof that can later be used as a proof that a certain event took place. These tokens are often necessary in order define a decision procedure. Of certain interest are **non-repudiation of origin** proving that a message originated from a certain entity and **non-repudiation of delivery** proving that a message has been delivered.

Payment system
Any system that can be used for payments.

Payment system provider
A party (such as a financial institution) that makes a payment system available.

Peer entity
Peer entities are two entities of the same layer but located on different devices. Usually, they interwork to provide a certain service.

Player
A player is a real-world person or body participating in the electronic marketplace, e.g. a buyer, a seller or a third party like a payment system provider, a notary service, etc.

Registration
A procedure by which a participant (usually a real person) registers his or her credentials with a so called registration authority. In return the participant gets a certificate as an electronic proof of the registration.

Revocation
The process of withdrawing the validity of a certificate, lost, or was registered by an impostor or that the user abused the granted rights.

Role
The function of an entity in a certain transaction. All transaction protocols are specified for roles, but performed by entities ‘playing’ the corresponding roles.
Roles are also needed for access control. The access rights usually depend on both the entity and its role.

Security attributes
Security attributes describes the security level that a certain transfer or exchange should provide. Examples are confidentiality, non-repudiation and anonymity.

Service
Services are offered by the joint functionality of a layer to its upper interface. One layer can offer several services, e.g. contract signing or fair exchange of goods against payment are both services of the exchange layer.

Service block
A service block consists of a manager and a number of modules (possibly 0).

Service manager
A service manager provides a common interface to the modules of the service blocks (the *external interface*), plus methods for negotiation and selection of an appropriate module. In addition the service manager provides services providing robustness against fault tolerance. The manager is always part of the *SEMPER* software.

Service module
The service module implements the services provided by the service block. It is expected that modules will be external (i.e., modules will only be implemented as part of *SEMPER* if necessary). The service module must support the *internal interface* in order to be used by the manager.

Session

A session is a set of primitives that operate on the same context, e.g. a business transaction. Sessions can be nested: a session on a higher layer may start several sessions on lower layers and exchange parts of its context with them.

Sub-transaction
A transaction which is invoked by an ongoing transaction.

Third party
A third party is an entity supporting a business transaction without being involved as business partner.

Transaction
A *SEMPER* transaction is a sequence of method invocations which should result in a state transition from one consistent state into another.

Transfer
The process of transferring any business items (payment, information, vouchers, etc.) from one player to another, usually in a secure way.
A transfer is realised by sending or receiving a container.

Type of module
A service block may support modules implementing the desired functionality in quite different ways and therefore the manager must support different internal interfaces. Modules supporting the same internal interfaces are said to be of the same type.

User
A user is anyone using the marketplace (e.g., buyer or seller).